

A LTL Fragment for GR(1)-Synthesis

Andreas Morgenstern and Klaus Schneider

University of Kaiserslautern
P.O. Box 3049
67653 Kaiserslautern, Germany
email: {morgenstern,schneider}@cs.uni-kl.de

The idea of automatic synthesis of reactive programs starting from temporal logic (LTL) specifications is quite old, but was commonly thought to be infeasible due to the known double exponential complexity of the problem. However, new ideas have recently renewed the interest in LTL synthesis: One major new contribution in this area is the recent work of Piterman et al. who showed how polynomial time synthesis can be achieved for a large class of LTL specifications that is expressive enough to cover many practical examples. These LTL specifications are equivalent to ω -automata having a so-called GR(1) acceptance condition. This approach has been used to automatically synthesize implementations of real-world applications. To this end, manually written deterministic ω -automata having GR(1) conditions were used instead of the original LTL specifications. However, manually generating deterministic monitors is, of course, a hard and error-prone task. In this paper, we therefore present algorithms to automatically translate specifications of a remarkable large fragment of LTL to deterministic monitors having a GR(1) acceptance condition so that the synthesis algorithms can start with more readable LTL specifications.

1 Introduction

In the last decades, the influence of computer systems on our everyday life has been constantly growing. As computer systems enter more and more safety-critical areas, their correctness is essentially important to avoid malfunctioning systems. Thus, one of the main challenges in computer science is the design of provably correct systems. Many of these safety-critical computer systems are reactive embedded systems. These are non-terminating systems that interact with their environments during their infinite computations. Typically, concurrency and infinite computations with respect to the environment make it difficult to analyze and design such systems correctly.

There are currently two main approaches to the design of provably correct reactive systems: In the first approach, called *formal verification*, one checks that a manually written implementation satisfies a given specification that is typically formulated in the temporal logic LTL [21, 9]. In the second approach, called LTL synthesis, a provably correct implementation is automatically derived from the given LTL specification. While formal verification is nowadays even routinely used in safety-critical system designs, LTL synthesis is still immature. Of course, the double exponential complexity of LTL synthesis compared to the single exponential one of LTL model checking is one reason for this situation. We believe, however, that the applicability of tools based on both methods can be significantly improved by better data structures and algorithms.

For example, a major breakthrough in formal verification has been achieved by symbolic representations of states and transitions with propositional formulas which became known as symbolic model checking [7]. With the advent of these succinct data structures and efficient decision procedures for propositional formulas, it has become possible to verify complex systems. In a similar way, new methods for SAT checking and SMT solvers opened the way to verify even larger systems.

It is natural to try to make use of such data structures and algorithms also for LTL synthesis. However, this is not directly possible, since the currently available LTL synthesis procedures consist of two steps: The first step is the translation of the LTL specification to an equivalent ω -automaton. The usual translation procedures generate a nondeterministic automaton that can be directly used for symbolic model checking. However, nondeterministic automata can, in general, not be used for LTL synthesis. Even though there are pseudo-deterministic automata like the good-for-games automata that can still be used for LTL synthesis, the second step usually consists of a determinization of the obtained automata (since deterministic automata can be definitely used without further restrictions). The problem is, however, that determinization is considerably more complex for ω -automata than for automata on finite words. In particular, a major drawback of the currently known determinization procedures is their explicit representation of the automata that does not make use of symbolic data structures. Since a translation from LTL to deterministic automata may lead to automata having a double exponential size in terms of the length of the formula, explicit state space representations are limited to handle very small LTL formulas.

One possibility to overcome the complexity problem of LTL synthesis is to consider restricted classes of LTL. For example, [1, 14] consider subsets of LTL to obtain deterministic automata with less than double exponential size. Wallmeier et al. [27] developed a synthesis algorithm to synthesize request-response specifications which are of the form $G(\phi_i \rightarrow F\psi_i)$ for multiple i which leads to a synthesis procedure with only exponential complexity. Piterman et al. proposed in [20] an approach to synthesize generalized reactivity formulas with rank 1 (abbreviated as GR(1) formulas), i.e. formulas of the form $(\bigwedge_{i=0}^N GF\phi_i) \rightarrow (\bigwedge_{j=0}^M GF\psi_j)$. Their algorithm runs in time K^3 where K is the size of the state space of the design. If a collection Φ_i of LTL formulas representing assumptions on the environment, and a collection Ψ_j of formulas representing conclusions for the system, can all be represented by deterministic Büchi automata, this approach can be used to obtain a synthesis procedure for the entire LTL specification $(\bigwedge_{i=0}^N \Phi_i) \rightarrow (\bigwedge_{j=0}^M \Psi_j)$.

The work reported in [20] has been extensively used. Its feasibility was demonstrated in [4, 5, 11] which considers ARM's Advance Micro-System Bus Architecture as well as a case study of a generalized buffer example included in IBM's RuleBase system. In those case studies, an implementation realizing the given formal specification has been derived and has been afterwards converted to a circuit. In fact, those case studies have been the first real-life blocks that have been automatically synthesized from high-level temporal logic specifications. Further applications include usage in the context of production of robot systems [28].

The main drawback of previously published works using the GR(1)-approach of Piterman et al. is that the unavoidable determinization step was carried out manually by a human developer, since no tool support for the translation of temporal logic formulas to corresponding ω -automata was available. The translation to deterministic automata is considerably hard in general [12] and may introduce errors due to the human intervention.

To eliminate this drawback from the GR(1)-approach, we present in this article a remarkable large subset of LTL that can be translated to sets of deterministic Büchi automata representing the assumptions on the environment and the guarantees a system has to satisfy. To this end, we reconsider the temporal logic hierarchy that has been investigated by Chang, Manna, Pnueli, Schneider and others [15, 8, 16, 17, 22, 23]. This temporal logic hierarchy defines subsets of LTL that correspond to the well-known automaton hierarchy, consisting of safety, guarantee/liveness, fairness/response/Büchi, persistence/co-Büchi properties as well as their boolean closures (obligation and reactivity properties). Using a syntactic characterization of this hierarchy [22, 23], we can, in particular, *syntactically* determine for given LTL formulas whether the formula can be represented by a deterministic Büchi automaton. Hence, given a set of formulas representing assumptions and conclusions, we can determine whether they can be used

as an input for GR(1)-synthesis. Clearly, since we only check this syntactically, it may be the case that we reject formulas that could be used for GR(1)-synthesis, but we never produce an error. In practice, it turned out that essentially no GR(1) formula is rejected by our syntactic check.

The syntactic approximation to determine GR(1) membership is one contribution of this paper. Another one is the observation that the negation of each formula that can be translated to a deterministic Büchi automaton can be translated to a non-deterministic co-Büchi automaton. It is well-known that non-deterministic co-Büchi automata can be determinized by the Breakpoint construction [18] that is well-suited for a symbolic implementation [19, 6]. From this co-Büchi automaton, we can easily obtain a deterministic Büchi automaton (again via negation, which is trivial for deterministic automata [23]) that is equivalent to the original formula. Hence, our second observation leads to a very efficient translation procedure for the identified LTL formulas to deterministic Büchi and co-Büchi automata.

We have implemented this synthesis procedure that (1) syntactically determines whether a formula can be represented with a GR(1)-property and (2) applies the mentioned symbolic determinization procedure for Büchi/co-Büchi automata. Finally, we apply the GR(1)-synthesis using an existing implementation of the GR(1)-Synthesis approach [3].

2 Preliminaries

2.1 Linear Temporal Logic LTL

For a given set of Boolean variables V , we define the set of LTL formulas by the following recursive definition:

Definition 1 (Syntax of Linear Temporal Logic (LTL)) *The set of LTL formulas over a set of variables V is the smallest set with the following properties:*

- $1, 0 \in \text{LTL}$
- $a \in \text{LTL}$ for $a \in V$
- *boolean operators:* $\neg\phi, \phi \wedge \psi, \phi \vee \psi \in \text{LTL}$ if $\phi, \psi \in \text{LTL}$
- *future temporal operators:* $X\phi, [\phi \underline{U} \psi], [\phi \text{ B } \psi]$ if $\phi, \psi \in \text{LTL}$
- *past temporal operators:* $\overleftarrow{X}\phi, \overleftarrow{X}\phi, [\phi \overline{U} \psi], [\phi \overline{\text{B}} \psi]$ if $\phi, \psi \in \text{LTL}$

The semantics of LTL can be given with respect to a path through a structure (e.g. an ω -automaton), where a path is an infinite word over the alphabet 2^V .

$X\phi$ holds on a path π at position t_0 if ϕ holds at position $t_0 + 1$ on the path. $[\phi \underline{U} \psi]$ holds at t_0 iff ψ holds for some position $\delta \geq t_0$ and ϕ holds invariantly for every position t with $t_0 \leq t < \delta$ i.e. ϕ holds *until* ψ holds. The *weak before* operator $[\phi \text{ B } \psi]$ holds at t_0 iff either ϕ holds before ψ becomes true for the first time after t_0 or ψ never holds after t_0 .

In addition to the future time temporal operators, there are also the corresponding past time temporal operators. These are defined analogously with the only difference that the direction of the flow of time is reversed. For example, $[\phi \overline{U} \psi]$ holds on a path at position t_0 iff there is a point of time δ with $\delta \leq t$ such that ψ holds on that path at position δ and ϕ holds for all positions t with $\delta < t \leq t_0$. The past time correspondence of the next-time operator is called the previous operator: $\overleftarrow{X}\phi$ holds on a path at position t_0 iff $t_0 > 0$ and ϕ holds at position $t_0 - 1$. Additionally, there is a weak variant, where $\overleftarrow{X}\phi$ holds on a path at position t_0 iff $t_0 = 0$ holds or ϕ holds at position $t_0 - 1$.

Other operators can be defined in terms of the above ones:

$$\begin{array}{ll}
G\varphi = [0 \text{ B } \neg\varphi] & \overleftarrow{G}\varphi = [0 \text{ } \overleftarrow{\text{B}} \neg\varphi] \\
F\varphi = [1 \text{ U } \varphi] & \overleftarrow{F}\varphi = [1 \text{ } \overleftarrow{\text{U}} \varphi] \\
[\varphi \text{ B } \psi] = \neg[\neg\varphi \text{ U } \psi] & [\varphi \text{ } \overleftarrow{\text{B}} \psi] = \neg[\neg\varphi \text{ } \overleftarrow{\text{U}} \psi] \\
[\varphi \text{ U } \psi] = [\psi \text{ B } (\neg\varphi \wedge \neg\psi)] & [\varphi \text{ } \overleftarrow{\text{U}} \psi] = [\psi \text{ } \overleftarrow{\text{B}} (\neg\varphi \wedge \neg\psi)] \\
[\varphi \text{ } \overleftarrow{\text{B}} \psi] = [\neg\psi \text{ U } (\varphi \wedge \neg\psi)] & [\varphi \text{ } \overleftarrow{\overleftarrow{\text{B}}} \psi] = [\neg\psi \text{ } \overleftarrow{\overleftarrow{\text{U}}} (\varphi \wedge \neg\psi)] \\
[\varphi \text{ W } \psi] = [(\varphi \wedge \psi) \text{ B } (\neg\varphi \wedge \neg\psi)] & [\varphi \text{ } \overleftarrow{\overleftarrow{\text{W}}} \psi] = [(\varphi \wedge \psi) \text{ } \overleftarrow{\overleftarrow{\text{B}}} (\neg\varphi \wedge \neg\psi)] \\
[\varphi \text{ } \overleftarrow{\overleftarrow{\text{U}}} \psi] = [\neg\psi \text{ U } (\varphi \wedge \neg\psi)] & [\varphi \text{ } \overleftarrow{\overleftarrow{\overleftarrow{\text{U}}}} \psi] = [\neg\psi \text{ } \overleftarrow{\overleftarrow{\overleftarrow{\text{U}}}} (\varphi \wedge \neg\psi)]
\end{array}$$

For example, $[\varphi \text{ U } \psi]$ is the *weak until* operator that can be alternatively defined as $[\varphi \text{ U } \psi] := [\varphi \text{ } \overleftarrow{\text{U}} \psi] \vee G\varphi$, i. e. the event ψ that is awaited for need not hold in the future. To distinguish weak and strong operators, the strong variants of a temporal operator are underlined in this paper (as done above).

2.2 ω -Automata

Definition 2 (ω -Automata) A ω -automaton $\mathfrak{A} = (\mathcal{Q}, \Sigma, \mathcal{I}, \mathcal{R}, \mathcal{A})$ over the alphabet Σ is given by a finite set of states \mathcal{Q} , a set \mathcal{I} of initial states, a transition relation $\mathcal{R} \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ and an acceptance condition $\mathcal{A} : \mathcal{Q}^\omega \rightarrow \{0, 1\}$.

Given an automaton $\mathfrak{A} = (\mathcal{Q}, \Sigma, \mathcal{I}, \mathcal{R}, \mathcal{A})$ and an infinite word $\alpha = a_0, a_1, \dots$ over Σ . Each infinite word $\beta = q_0, q_1, \dots$ with $q_0 \in \mathcal{I}$ and $q_{i+1} \in \delta(q_i, \alpha_i)$ for $i \geq 0$ is called a run of α through \mathfrak{A} . The run is *accepting* if $\mathcal{A}(\beta) = 1$. We say that \mathfrak{A} accepts α whenever an accepting run of α through \mathfrak{A} exists.

Using standard terminology, we say that \mathfrak{A} is *deterministic*, if exactly one initial state exists and for each $q \in \mathcal{Q}$ and each input $\sigma \in \Sigma$ there exists exactly one $s' \in \mathcal{Q}$ with $(s, \sigma, s') \in \mathcal{R}$. In that case we write $\mathfrak{A} = (\mathcal{Q}, \Sigma, q_0, \delta, \mathcal{A})$ with an initial state q_0 and a deterministic transition function $\delta : \mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$.

In the following, we assume that $\mathcal{Q} = 2^V$ for a set V of state variables. Moreover, we assume sets X and Y of input and output variables that form the inputs $\mathcal{X} = 2^X$ and outputs $\mathcal{Y} = 2^Y$ of the system such that $\Sigma = \mathcal{X} \times \mathcal{Y}$. Having this view, we define a state set \mathcal{Q}_φ to contain exactly those states where the propositional encoding of the state variables V satisfy φ . Thus, we can conveniently define acceptance conditions by LTL specifications.

2.3 Classical Acceptance Conditions

In the past, several kinds of acceptance conditions have been proposed and their different expressiveness have been studied in depth. In particular, the following acceptance conditions have been considered [26, 25, 23].

- A run is accepted by a safety condition $G\varphi$ if the run exclusively runs through the set \mathcal{Q}_φ .
- A run is accepted by a liveness condition $F\varphi$ if the run visits at least one state of the set \mathcal{Q}_φ at least once.
- A run is accepted by a prefix¹ condition $\bigwedge_i (G\varphi_i \vee F\psi_i)$ if for all i either the run exclusively runs through the set \mathcal{Q}_{φ_i} or visits \mathcal{Q}_{ψ_i} at least once.

¹These conditions are also called Staiger-Wagner or obligation conditions.

- A run is accepted by a Büchi condition $\text{GF}\phi$ if the run visits at least one state of the set \mathcal{Q}_ϕ infinitely often.
- A run is accepted by a co-Büchi condition $\text{FG}\phi$ if the run visits only states of the set \mathcal{Q}_ϕ infinitely often.
- Finally, a run is accepted by a Streett (or reactivity) condition $\bigwedge_{i=0}^f \text{GF}\phi_i \vee \text{FG}\psi_i$ if for all i either the run visits at least one state from \mathcal{Q}_{ϕ_i} or the run visits only states of the set \mathcal{Q}_{ψ_i} infinitely often.

2.4 GR(1)-Specifications for LTL Synthesis

The task of LTL synthesis is to develop a system that controls the output variables Y so that no matter how the environment chooses the input variables X , a LTL specification is satisfied. Thus, instead of using one of the classical acceptance conditions, it is more convenient for synthesis to consider specifications of the form $\phi \rightarrow \psi$ where ϕ represents assumptions on the environment and ψ represents conclusions/guarantees the system has to satisfy. In particular, Generalized Reactivity (1) acceptance [4, 5, 11, 20] attracted some interest in the community: here the assumptions and guarantees are all Büchi conditions, i. e. we seek a system satisfying the following acceptance condition:

$$\text{GR}(1) := \left(\bigwedge_{i=1}^n \text{GF}p_i \right) \rightarrow \left(\bigwedge_{j=1}^m \text{GF}q_j \right) \quad (1)$$

The class of specifications to which the algorithms of [4, 5, 11, 20] can be applied is much more general than the limited form presented in equation 1: The algorithm can be applied to any specification of the form $(\bigwedge_{i=1}^n \phi_i) \rightarrow (\bigwedge_{j=1}^m \psi_j)$ where each ϕ_i, ψ_j is specified by a deterministic Büchi automaton.

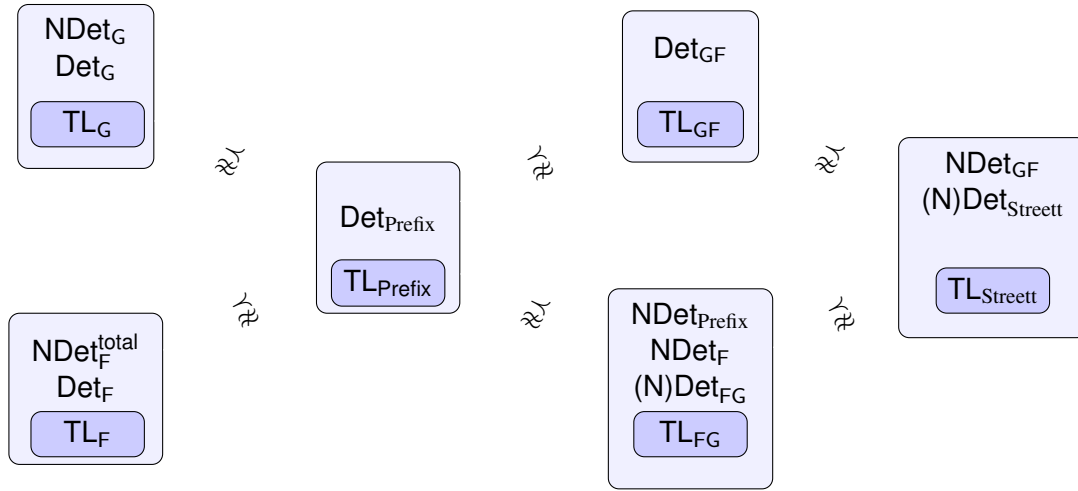
Definition 3 ([13]) Assume we are given n deterministic Büchi automata $\mathfrak{A}_1^a, \dots, \mathfrak{A}_n^a$ for the environment's assumptions and m deterministic Büchi automata $\mathfrak{A}_1^g, \dots, \mathfrak{A}_m^g$ for the system's guarantees with $\mathfrak{A}_i^a = (\mathcal{Q}_i^a, \Sigma, q_{0,i}^a, \delta_i^a, \text{GF}p_i)$ and $\mathfrak{A}_j^g = (\mathcal{Q}_j^g, \Sigma, q_{0,j}^g, \delta_j^g, \text{GF}q_j)$. Then, we define an automaton $\mathfrak{A}^{\text{GR}(1)} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{A})$ as the product of all automata \mathfrak{A}_i^a and \mathfrak{A}_j^g where the state space is $\mathcal{Q} = \mathcal{Q}_1^a \times \dots \times \mathcal{Q}_n^a \times \mathcal{Q}_1^g \times \dots \times \mathcal{Q}_m^g$, the transition function is $\delta((q_1^a, \dots, q_m^g), \sigma) = (\delta_1^a(q_1^a, \sigma), \dots, \delta_m^g(q_m^g, \sigma))$ and the initial state is $q_0 = (q_{0,1}^a, \dots, q_{0,m}^g)$. The acceptance condition $\mathcal{A} = (\bigwedge_{i=1}^n \text{GF}p_i) \rightarrow (\bigwedge_{j=1}^m \text{GF}q_j)$ is a GR(1) condition.

Thus, a run of $\mathfrak{A}^{\text{GR}(1)}$ is accepting if either all sets \mathcal{Q}_{q_j} are visited infinitely often or at least some set \mathcal{Q}_{p_i} is visited only finitely often.

2.5 Games

A game $\mathfrak{G} = (\mathcal{Q}, \Sigma, q_0, \delta, \mathcal{A})$ is a deterministic ω -automaton with an input alphabet $\Sigma = \mathcal{X} \times \mathcal{Y}$. A play of \mathfrak{G} is an infinite sequence of states $\pi = q_0 q_1 q_2 \dots \in \mathcal{Q}^\omega$ where $q_{i+1} = \delta(q_i, \sigma_i)$ for $i \geq 0$. The letters $\sigma_i = (x_i, y_i)$ are successively chosen by the players: in each step, the environment first chooses x_i , and then the system chooses y_i . A play π is won by the system if $\mathcal{A}(\pi) = 1$. Otherwise, the game is won by the environment. Note that the environment cannot react to the outputs generated by the system and thus acts like a Moore machine. In contrast, the system we would like to synthesize acts like a Mealy machine.

We solve the game, attempting to decide whether the game is winning for the environment or the system. If the environment is winning, the specification is unrealizable. If the system is winning, we

Figure 1: (Borel) Hierarchy of ω -Automata and Temporal Logic

synthesize a winning strategy (which is essentially a Mealy automaton) using the algorithms given in [4, 5, 11, 20].

Previous works regarding the synthesis with respect to GR(1)-synthesis had to manually generate the deterministic automata. In this paper, we show how to automatically obtain deterministic Büchi automata from a fragment of LTL using the well-known Breakpoint construction. This fragment of LTL is a natural fragment of LTL embedded in the well-known temporal-logic hierarchy [15, 8, 16, 17, 22, 23].

3 Temporal Logic vs. Automaton Hierarchy

3.1 The Automaton Hierarchy

The classical acceptance conditions, i.e., safety, guarantee/liveness, fairness/response/Büchi, persistence/co-Büchi properties, define the corresponding automaton classes $(N)Det_G$, $(N)Det_F$, $(N)Det_{GF}$, and $(N)Det_{FG}$, respectively. Moreover, their boolean closures can be represented by the automaton classes $(N)Det_{Prefix}$ and $(N)Det_{Streett}$ whose acceptance conditions have the forms $\bigwedge_{j=0}^f G\phi_j \vee F\psi_j$ and $\bigwedge_{j=0}^f GF\phi_j \vee FG\psi_j$, respectively.

The expressiveness of these classes is illustrated in Figure 1, where $\mathcal{C}_1 \lesssim \mathcal{C}_2$ means that for any automaton in \mathcal{C}_1 , there is an equivalent one in \mathcal{C}_2 . Moreover, we define $\mathcal{C}_1 \approx \mathcal{C}_2 := \mathcal{C}_1 \lesssim \mathcal{C}_2 \wedge \mathcal{C}_2 \lesssim \mathcal{C}_1$ and $\mathcal{C}_1 \not\approx \mathcal{C}_2 := \mathcal{C}_1 \lesssim \mathcal{C}_2 \wedge \neg(\mathcal{C}_1 \approx \mathcal{C}_2)$. As can be seen, the hierarchy consists of six different classes, and each class has a deterministic representative.

3.2 The Temporal Logic Hierarchy

In [8, 22, 23], corresponding hierarchies for temporal logics have been defined. Following [22, 23], we define the hierarchy of temporal logic formulas syntactically by the grammar rules of Fig. 2:

Definition 4 (Temporal Logic Classes) For $\kappa \in \{G, F, Prefix, FG, GF, Streett\}$, we define the logics TL_κ by the grammars given in Fig. 2, where TL_κ is the set of formulas that can be derived from the nonterminal P_κ (V_Σ represents any variable $v \in V_\Sigma$).

$P_G ::= V_\Sigma \mid \neg P_F \mid P_G \wedge P_G \mid P_G \vee P_G$ $\mid \bigwedge P_G \mid [P_G \bigcup P_G]$ $\mid \bigwedge P_G \mid [P_G \bigcup P_G]$ $\mid \bigwedge P_G \mid [P_G \bigcup P_G]$ $\mid \bigwedge P_G \mid [P_G \bigcup P_G]$	$P_F ::= V_\Sigma \mid \neg P_G \mid P_F \wedge P_F \mid P_F \vee P_F$ $\mid \bigwedge P_F \mid [P_F \bigcup P_F]$ $\mid \bigwedge P_F \mid [P_F \bigcup P_F]$ $\mid \bigwedge P_F \mid [P_F \bigcup P_F]$ $\mid \bigwedge P_F \mid [P_F \bigcup P_F]$
$P_{\text{Prefix}} ::= P_G \mid P_F \mid \neg P_{\text{Prefix}} \mid P_{\text{Prefix}} \wedge P_{\text{Prefix}} \mid P_{\text{Prefix}} \vee P_{\text{Prefix}}$	
$P_{GF} ::= P_{\text{Prefix}}$ $\mid \neg P_{FG} \mid P_{GF} \wedge P_{GF} \mid P_{GF} \vee P_{GF}$ $\mid \bigwedge P_{GF} \mid \bigwedge P_{GF} \mid \bigwedge P_{GF}$ $\mid [P_{GF} \bigcup P_{GF}] \mid [P_{GF} \bigcup P_{GF}]$ $\mid [P_{GF} \bigcup P_{GF}] \mid [P_{GF} \bigcup P_{GF}]$	$P_{FG} ::= P_{\text{Prefix}}$ $\mid \neg P_{GF} \mid P_{FG} \wedge P_{FG} \mid P_{FG} \vee P_{FG}$ $\mid \bigwedge P_{FG} \mid \bigwedge P_{FG} \mid \bigwedge P_{FG}$ $\mid [P_{FG} \bigcup P_{FG}] \mid [P_{FG} \bigcup P_{FG}]$ $\mid [P_{FG} \bigcup P_{FG}] \mid [P_{FG} \bigcup P_{FG}]$
$P_{\text{Streett}} ::= P_{GF} \mid P_{FG} \mid \neg P_{\text{Streett}} \mid P_{\text{Streett}} \wedge P_{\text{Streett}} \mid P_{\text{Streett}} \vee P_{\text{Streett}}$	

Figure 2: Syntactic Characterizations of the Classes of the Temporal Logic Hierarchy

Typical safety conditions like $G\phi$ or $G[a \cup b]$ that state that something bad never happens, are contained in TL_G . Liveness conditions like $F\phi$ are contained in TL_F . Finally, fairness conditions like $GF\phi$ that demand that something good infinitely often happens, are contained in TL_{GF} while stabilization/persistence properties like $FG\phi$ that demand that after a finite interval, nothing bad happens are contained in TL_{FG} .

3.3 Relating the Temporal Logic and the Automata Hierarchy

In [22, 23] several translation procedures are given to translate formulas from TL_κ to equivalent $(N)Det_\kappa$ automata. In particular, the following is an important result:

Theorem 1 (Temporal Logic and Automaton Hierarchy) *Given a formula $\Phi \in TL_\kappa$, we can construct a deterministic ω -automaton $\mathfrak{A} = (2^Q, \mathcal{I}, \mathcal{R}, \lambda, \mathcal{A})$ of the class Det_κ in time $O(2^{|\Phi|})$ with $|Q| \leq 2^{|\Phi|}$ state variables. Therefore, $\mathfrak{A} = (2^Q, \mathcal{I}, \mathcal{R}, \lambda, \mathcal{A})$ is a symbolic representation of a deterministic automaton with $O(2^{2^{|\Phi|}})$ states.*

The above results are already proved in detail in [23], where translation procedures from TL_κ to $NDet_\kappa$ have been constructed. Moreover, it has been shown in [23] that the subset construction can be used to determinize the automata that stem from the classes TL_G and TL_F and that the Miyano-Hayashi breakpoint construction is sufficient to determinize the automata that stem from the translation of formulas from TL_{FG} and TL_{GF} . Since TL_{Prefix} and TL_{Streett} are the boolean closures of $TL_G \cup TL_F$ and $TL_{FG} \cup TL_{GF}$, respectively, the remaining results for TL_{Prefix} and TL_{Streett} follow from the boolean combinations of Det_G/Det_F and Det_{FG}/Det_{GF} , respectively.

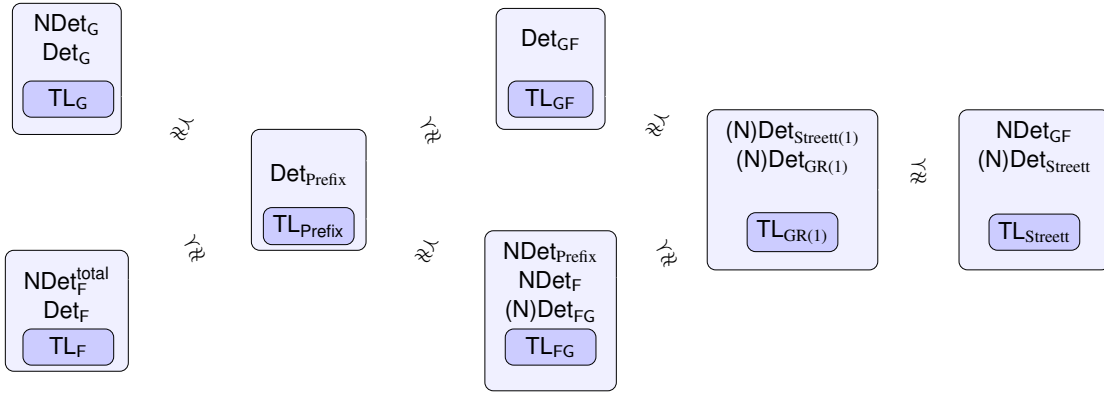
The final step consists of computing the boolean closure of the acceptance conditions. To this end, it is shown in [23] how arbitrary boolean combinations of $G\phi$ and $F\phi$ with propositional formulas ϕ are translated to equivalent Det_{Prefix} automata, and analogously, how arbitrary boolean combinations of $GF\phi$ and $FG\phi$ with propositional formulas ϕ are translated to equivalent Det_{Streett} automata.

4 A LTL Fragment for GR(1)-Synthesis

Using the previously mentioned temporal logic hierarchy, we define a fragment of LTL that can be easily translated to a set of deterministic Büchi automata for the assumptions and a set of deterministic Büchi automata for the guarantees (Figure 3).

$P_G ::= V_\Sigma \mid \neg P_F \mid P_G \wedge P_G \mid P_G \vee P_G$ $\mid \bigwedge P_G \mid [P_G \bigvee P_G]$ $\mid \bigwedge P_G \mid [P_G \bigvee P_G]$ $\mid \bigwedge P_G \mid [P_G \bigvee P_G]$	$P_F ::= V_\Sigma \mid \neg P_G \mid P_F \wedge P_F \mid P_F \vee P_F$ $\mid \bigwedge P_F \mid [P_F \bigvee P_F]$ $\mid \bigwedge P_F \mid [P_F \bigvee P_F]$ $\mid \bigwedge P_F \mid [P_F \bigvee P_F]$
$P_{\text{Prefix}} ::= P_G \mid P_F \mid \neg P_{\text{Prefix}} \mid P_{\text{Prefix}} \wedge P_{\text{Prefix}} \mid P_{\text{Prefix}} \vee P_{\text{Prefix}}$	
$P_{GF} ::= P_{\text{Prefix}}$ $\mid \neg P_{FG} \mid P_{GF} \wedge P_{GF} \mid P_{GF} \vee P_{GF}$ $\mid \bigwedge P_{GF} \mid \bigwedge P_{GF} \mid \bigwedge P_{GF}$ $\mid [P_{GF} \bigvee P_{GF}] \mid [P_{GF} \bigvee P_{GF}]$ $\mid [P_{GF} \bigvee P_{GF}] \mid [P_{GF} \bigvee P_{GF}]$	$P_{FG} ::= P_{\text{Prefix}}$ $\mid \neg P_{GF} \mid P_{FG} \wedge P_{FG} \mid P_{FG} \vee P_{FG}$ $\mid \bigwedge P_{FG} \mid \bigwedge P_{FG} \mid \bigwedge P_{FG}$ $\mid [P_{FG} \bigvee P_{FG}] \mid [P_{FG} \bigvee P_{FG}]$ $\mid [P_{FG} \bigvee P_{FG}] \mid [P_{FG} \bigvee P_{FG}]$
$P_{\text{Assume}} ::= P_{GF} \mid P_{\text{Assume}} \wedge P_{\text{Assume}}$	$P_{\text{Guarantee}} ::= P_{GF} \mid P_{\text{Guarantee}} \wedge P_{\text{Guarantee}}$
$P_{\text{GR}(1)} ::= P_{\text{Assume}} \rightarrow P_{\text{Assert}}$	

Figure 3: A LTL Fragment for GR(1)-Synthesis

Figure 4: (Borel) Hierarchy of ω -Automata and Temporal Logic with GR(1)

As can be seen, our LTL fragment is naturally embedded in the temporal logic hierarchy. The formulas that syntactically belong to our LTL fragment are those formulas that are derived from the nonterminal $P_{\text{GR}(1)}$, thus, these are implications of formulas that are derived from the nonterminals P_{Assume} and P_{Assert} , respectively, which are both conjunctions of TL_{GF} -formulas.

Concerning the automata hierarchy, we can translate these formulas to automata with a GR(1)-acceptance condition, i.e. a generalization of a Streett(1) condition. In [2], it is shown that a GR(1)-condition can be equivalently expressed by a Streett(1)-condition, i.e. a Streett condition with only one acceptance pair. Hence, we obtain the "enriched" automata hierarchy shown in Figure 4 together with the following corollary that easily follows from Theorem 1:

Corollary 1 *Given a $P_{\text{GR}(1)}$ -formula of the form $\Phi = (\varphi_1 \wedge \dots \wedge \varphi_n) \rightarrow (\psi_1 \wedge \dots \wedge \psi_m)$, we can compute n deterministic Büchi automata $\mathcal{A}_{\varphi_1}^a, \dots, \mathcal{A}_{\varphi_n}^a$ and m deterministic Büchi automata $\mathcal{A}_{\psi_1}^s, \dots, \mathcal{A}_{\psi_m}^s$ such that \mathcal{A}_{φ_i} (\mathcal{A}_{ψ_j}) is initially equivalent to φ_i (resp. ψ_j). Hence the GR(1)-automaton obtained from those automata according to Definition 3 is initially equivalent to Φ .*

5 Experiments

In our previous work, we had already implemented a toolset Averest [24] whose inputs are programs written in the Esterel-like synchronous programming language Quartz [24]. Averest compiles the synchronous programs to guarded actions which can be used in turn to generate sequential and concurrent software, hardware or symbolic transition relations for formal verification. Specifications can be given in various temporal logics and the μ -calculus. Averest provides a lot of translations from temporal logic to either ω -automata or directly to the μ -calculus (see [23] for these translations).

For this paper, we implemented an additional tool Quartz2Marduk that takes as input a set of LTL formulas that represent assumptions and assertions/guarantees of a GR(1) specification (see example shown in Figure 5). We then check whether these specifications belong to the class that can be used for GR(1)-synthesis. If so, we automatically generate deterministic automata that are equivalent to the specification. The automata are automatically minimized using a form of delayed simulation [10] and are afterwards used to generate a file as input to the Marduk² tool [3]. Marduk is a re-implementation of Anzu [11] with some new features. It is basically a BDD-based implementation of the algorithm given in [20].

Included with Marduk came two case studies that are described in [4, 5, 11]. The first case study is the GenBuf example that is used as a tutorial in IBM's RuleBase system. The second example is ARM's *Advanced Microcontroller Bus Architecture (AMBA)* which defines the *Advanced High performance Bus (AHB)*, an on-chip communication standard that connects devices like processor cores, caches and DMA arbiters.

In [4, 5, 11] temporal logic specifications for those case studies are given along with some hints how deterministic automata for these specifications can be manually obtained. Marduk came with an input file that already contained those manually generated deterministic automata. In our tool, all we had to do is to simply write down the temporal logic specifications given in [4, 5, 11] and compile it to a Marduk input file.

After having compiled the Marduk input files, we ran Marduk with dynamic variable ordering enabled, leaving the other options untouched. The results of our experiments is given in table 6. The first column given there is the name of the case study, the second column is the time (in seconds) our tool needed to perform determinization. The third column lists the number of state variables that were generated by our tool and the manually generated deterministic automata. The next column lists the number of BDD Nodes for the generated strategy. Finally, the last column lists the runtime of Marduk for the automatically generated automata and the respective time for the manually generated automata. In the table, TO means that the synthesis procedure could not be finished within 50000 seconds³.

6 Discussion

The GR(1)-approach is one of the most successful approaches to LTL synthesis today [4, 5, 11] that has already found applications apart from its primary target [28]. One interesting question regarding the GR(1)-synthesis approach is its good algorithmic behavior of having a cubic runtime despite the fact that many specifications can be rewritten to a deterministic automaton having a GR(1)-acceptance

²Actually, our current implementation generates an Anzu [11] file and we use a tool included with Marduk to translate this Anzu file to a Marduk file.

³We can not satisfactorily explain why the synthesis for the AMBA model needed more time for 6 masters than for 7 masters using our determinization procedure. However, the same holds for the manually generated automata where this observation can be done for 8 respectively for 9 masters. However, a similar observation was also reported in [5].

```

macro N() = 4;

module GenBuf(bool[N()+1] ?BtoS_Ack, bool[2] ?BtoR_Req,
              bool[N()+1] !StoB_Req, bool[2] !RtoB_Ack) {
} satisfies {
  spec_1: assert
    A (forall(i=0..N())
      G (StoB_Req[i] -> F BtoS_Ack[i]));
  spec_2: assert
    A (forall(i=0..N())
      G (!StoB_Req[i] & X StoB_Req[i] -> X !BtoS_Ack[i]));
  spec_3: assert
    A (forall(i=0..N())
      G (StoB_Req[i] & ! BtoS_Ack[i] -> X StoB_Req[i]));
  spec_4: assert
    A (forall(i=0..N())
      G (BtoS_Ack[i] -> X !StoB_Req[i]));
}

```

Figure 5: An Example Quartz File with a GR(1) Specification having only Assertions

condition. This question has been answered in [2] where it is shown that in fact an automaton with GR(1)-acceptance condition is equivalent to a Streett automaton having only one acceptance pair.

In this article, we gave the corresponding temporal logic view: We presented a fragment of LTL that is ‘naturally’ embedded in the temporal logic hierarchy and that can be easily translated to a corresponding deterministic GR(1)-automaton. We have implemented a tool that is able to translate any formula from this fragment to a corresponding deterministic GR(1)-automaton. This is a useful improvement in the expressivity and usage of the GR(1)-approach: instead of having the need to generate deterministic automata manually, the input to our tool is a more readable LTL formula.

However, this higher expressivity comes to a cost: Not too surprisingly, running Marduk on the manually generated automata took a significant smaller amount of time than on the automatically generated automata and moreover, generated smaller BDDs for the strategies. However, the manually generated automata have undergone heavy (hand-crafted) minimization steps⁴ and hence we expect that further improvements on the determinization or the minimization step of our tool could also significantly improve our results.

7 Acknowledgements

We would like to thank Georg Hofferek for his kind help with the tool Marduk.

⁴Compare the difference in the runtime of the Anzu tool reported in [4] with the one reported in [5].

Model	Det (s)	State Vars		Strategy Nodes		Solve(t)	
		Auto	Manu	Auto	Manu	Auto	Manu
GenBuf 2	0.1	12	3	8.755	3.344	0.86	0.25
GenBuf 3	0.1	12	3	19.087	4.237	1.96	0.3
GenBuf 4	0.2	12	3	25.653	5.546	2.12	0.63
GenBuf 5	0.2	12	3	39.356	11.916	12.88	1.34
GenBuf6	0.3	12	3	26.139	15.605	5.61	2.38
GenBuf7	0.3	12	3	117.625	18.894	41.92	3.75
GenBuf8	0.3	12	3	45.238	24.302	11.24	5.14
GenBuf9	0.3	12	3	27.507	24.493	12.7	7.8
GenBuf10	0.3	12	3	67.879	51.605	44.91	25.3
Amba2	0.6	9	7	38.107	50.816	3.0	1.97
Amba3	1.1	10	8	77.033	122.027	14.4	10.64
Amba4	1.8	11	9	451.456	503.622	66.9	98.32
Amba5	7.2	12	10	1.194.190	825.294	1221.7	381.34
Amba6	19.4	13	11	4.929.635	989.482	46815	420.96
Amba7	42.0	14	12	2.052.871	1.037.608	4555.2	904.78
Amba8	83.1	15	13	TO	3.625.518	TO	13617.19
Amba9	403.6	16	14	TO	1.331.441	TO	4215.94
Amba10	580.16	17	15	TO	3.034.060	TO	7325.85

Figure 6: Experimental Results

References

- [1] R. Alur & S. La Torre (2004): *Deterministic Generators and Games for LTL Fragments*. *ACM Transactions on Computational Logic (TOCL)* 5(1), pp. 1–15, doi:10.1145/963927.963928.
- [2] R. Bloem, K. Chatterjee, K. Greimel, T.A. Henzinger & B. Jobstmann (2010): *Robustness in the Presence of Liveness*. In T. Touili, B. Cook & P. Jackson, editors: *Computer Aided Verification (CAV)*. LNCS 6174, Springer, Edinburgh, UK, pp. 410–424, doi:10.1007/978-3-642-14295-6_36.
- [3] R. Bloem, A. Cimatti, K. Greimel, G. Hofferek, R. Könighofer, M. Roveri, V. Schuppan & R. Seeber (2010): *RATSY - A New Requirements Analysis Tool with Synthesis*. In T. Touili, B. Cook & P. Jackson, editors: *Computer Aided Verification (CAV)*. LNCS 6174, Springer, Edinburgh, UK, pp. 425–429, doi:10.1007/978-3-642-14295-6.
- [4] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli & M. Weiglhofer (2007): *Automatic hardware synthesis from specifications: a case study*. In R. Lauwereins & J. Madsen, editors: *Design, Automation and Test in Europe (DATE)*. IEEE Computer Society, Nice, France, pp. 1188–1193.
- [5] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli & M. Weiglhofer (2007): *Specify, Compile, Run: Hardware from PSL*. *Electronic Notes in Theoretical Computer Science (ENTCS)* 190, pp. 3–16, doi:10.1016/j.entcs.2007.09.004.
- [6] U. Boker & O. Kupferman (2009): *Co-ing Büchi Made Tight and Useful*. In: *Logic in Computer Science (LICS)*. IEEE Computer Society, Los Angeles, California, USA, pp. 245–254, doi:10.1109/LICS.2009.32.
- [7] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill & L.J. Hwang (1990): *Symbolic Model Checking: 10²⁰ States and Beyond*. In: *Logic in Computer Science (LICS)*. IEEE Computer Society, Washington, DC, USA, pp. 1–33, doi:10.1109/LICS.1990.113767.

- [8] E.Y. Chang, Z. Manna & A. Pnueli (1992): *Characterization of Temporal Property Classes*. In W. Kuich, editor: *International Colloquium on Automata, Languages and Programming (ICALP)*. LNCS 623, Springer, Vienna, Austria, pp. 474–486.
- [9] E.A. Emerson (1990): *Temporal and Modal Logic*. In J. van Leeuwen, editor: *Handbook of Theoretical Computer Science*, chapter 16. B: Formal Models and Semantics, Elsevier, pp. 995–1072.
- [10] C. Fritz (2005): *Simulation-Based Simplification of omega-Automata*. Ph.D. thesis, Technischen Fakultät der Christian-Albrechts-Universität zu Kiel, Germany.
- [11] B. Jobstmann, S. Galler, M. Weiglhofer & R. Bloem (2007): *Anzu: A Tool for Property Synthesis*. In W. Damm & H. Hermanns, editors: *Computer Aided Verification (CAV)*. LNCS 4590, Springer, Berlin, Germany, pp. 258–262, doi:10.1007/978-3-540-73368-3_29.
- [12] O. Kupferman & M.Y. Vardi (1998): *Freedom, Weakness, and Determinism: From Linear-Time to Branching-Time*. In: *Logic in Computer Science (LICS)*. IEEE Computer Society, Indianapolis, Indiana, USA, pp. 81–92, doi:10.1109/LICS.1998.705645.
- [13] R. Könighofer, G. Hofferek & R. Bloem (2009): *Debugging formal specifications using simple counterstrategies*. In: *Formal Methods in Computer-Aided Design (FMCAD)*. IEEE Computer Society, Austin, Texas, USA, pp. 152–159, doi:10.1109/FMCAD.2009.5351127.
- [14] M. Maidl (2000): *The Common Fragment of CTL and LTL*. In: *Foundations of Computer Science (FOCS)*. pp. 643–652.
- [15] Z. Manna & A. Pnueli (1987): *A Hierarchy of Temporal Properties*. In: *Principles of Distributed Computing (PODC)*. p. 205, doi:10.1145/41840.41857.
- [16] Z. Manna & A. Pnueli (1990): *A hierarchy of temporal properties*. In: *Principles of Distributed Computing (PODC)*. ACM, Quebec City, Quebec, Canada, pp. 377–408.
- [17] Z. Manna & A. Pnueli (1991): *Completing the temporal picture*. *Theoretical Computer Science (TCS)* 83(1), pp. 97–130, doi:10.1016/0304-3975(91)90041-Y.
- [18] S. Miyano & T. Hayashi (1984): *Alternating automata on ω -words*. *Theoretical Computer Science (TCS)* 32, pp. 321–330, doi:10.1016/0304-3975(84)90049-5.
- [19] A. Morgenstern, K. Schneider & S. Lamberti (2008): *Generating Deterministic ω -Automata for most LTL Formulas by the Breakpoint Construction*. In C. Scholl & S. Disch, editors: *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*. Shaker, Freiburg, Germany, pp. 119–128.
- [20] N. Piterman, A. Pnueli & Y. Sa’ar (2006): *Synthesis of Reactive(1) Designs*. In E.A. Emerson & K.S. Namjoshi, editors: *Verification, Model Checking, and Abstract Interpretation (VMCAI)*. LNCS 3855, Springer, Charleston, South Carolina, USA, pp. 364–380, doi:10.1007/11609773_24.
- [21] A. Pnueli (1977): *The Temporal Logic of Programs*. In: *Foundations of Computer Science (FOCS)*. IEEE Computer Society, Providence, Rhode Island, USA, pp. 46–57, doi:10.1109/SFCS.1977.32.
- [22] K. Schneider (2001): *Improving Automata Generation for Linear Temporal Logic by Considering the Automata Hierarchy*. In R. Nieuwenhuis & A. Voronkov, editors: *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*. LNAI 2250, Springer, Havana, Cuba, pp. 39–54, doi:10.1007/3-540-45653-8_3.
- [23] K. Schneider (2003): *Verification of Reactive Systems - Formal Methods and Algorithms*. Texts in Theoretical Computer Science (EATCS Series), Springer.
- [24] K. Schneider (2009): *The Synchronous Programming Language Quartz*. Internal Report 375, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany.
- [25] W. Thomas (1990): *Automata on Infinite Objects*. In J. van Leeuwen, editor: *Handbook of Theoretical Computer Science*, chapter 4. B: Formal Models and Semantics, Elsevier, pp. 133–191.
- [26] K. Wagner (1979): *On ω -regular sets*. *Information and Control* 43(2), pp. 123–177.

- [27] N. Wallmeier, P. Hütten & W. Thomas (2003): *Symbolic Synthesis of Finite-State Controllers for Request-Response Specifications*. In O.H. Ibarra & Z. Dang, editors: *Conference on Implementation and Application of Automata (CIAA)*. LNCS 2759, Springer, Santa Barbara, California, USA, pp. 11–22, doi:10.1007/3-540-45089-0_3.
- [28] T. Wongpiromsarn, U. Topcu & R.M. Murray (2010): *Receding horizon control for temporal logic specifications*. In K.H. Johansson & W. Yi, editors: *Hybrid Systems: Computation and Control (HSCC)*. ACM, Stockholm, Sweden, pp. 101–110, doi:10.1145/1755952.1755968.